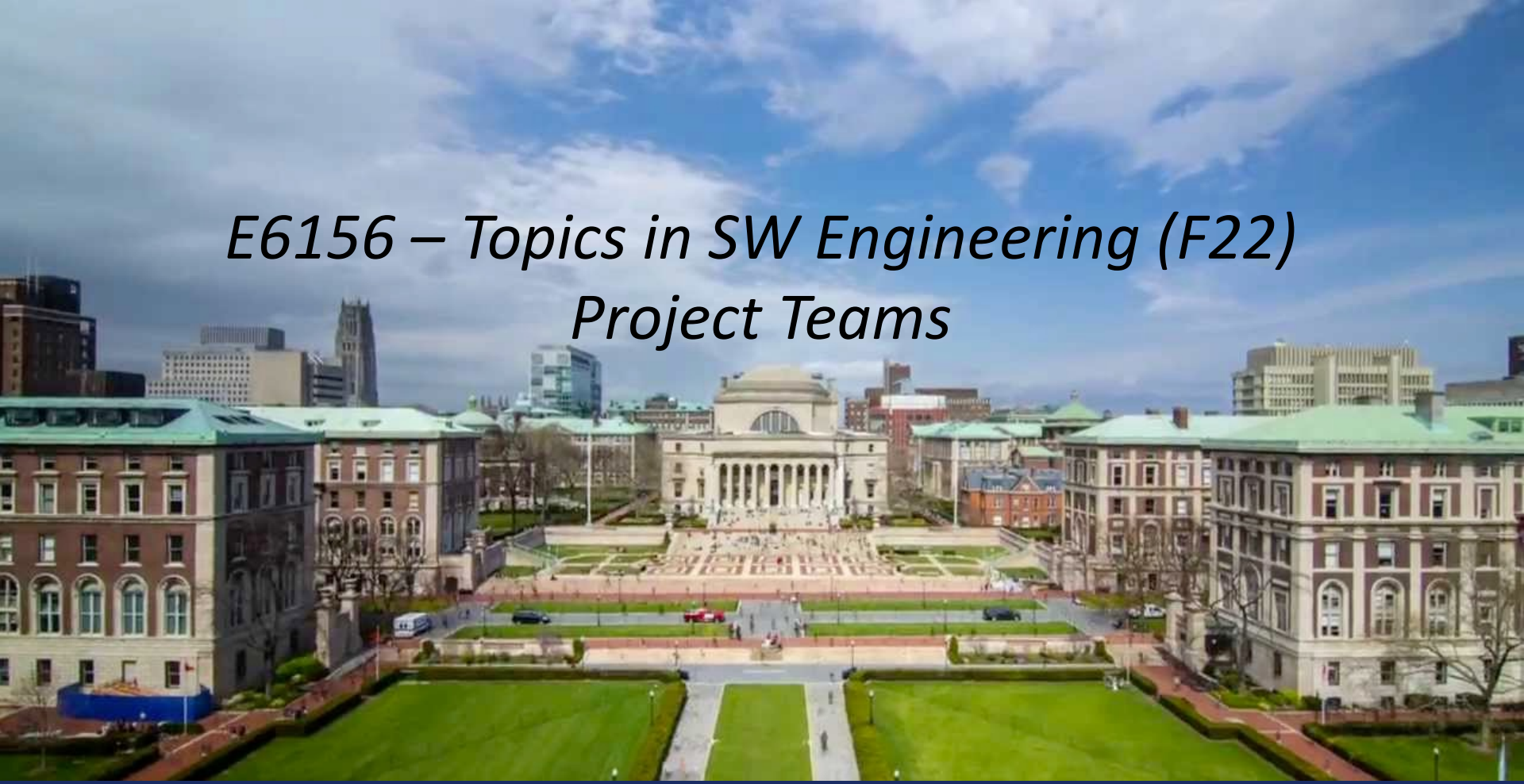
E6156 – Topics in SW Engineering (F22)

Cloud Computing

Lecture 1: Introduction



E6156 – Topics in SW Engineering (F22)
Project Teams

E6156 – Topics in SW Engineering: Cloud Computing

Faculty do not manage waitlists for some courses, including W4111. The academic admin staff in the CS Department manages the waitlist, priorities and enrollment.

Today's Contents and Agenda

Contents and Agenda

- Introduction
  - Your instructor
  - Course logistics, objectives, grading and overview
- Overview of core concepts
  - Cloud computing
  - Microservices
  - Full-stack web application
  - Virtual Machines, IaaS
- Course first steps
  - Form teams
  - Individual assignment – build and deploy ”Hello world!” application
  - Starting example walkthrough

Introduction

About your Instructor

- 35 years in computer science industry:
  - IBM Fellow.
  - Microsoft Technical Fellow.
  - Chief Technology Officer, CA technologies.
  - Dell Senior Technical Fellow.
  - CTO, Co-Founder, Seeka.tv.
  - Ansys (current):
    - Ansys Fellow, Chief SW Architect;
    - Lead development on cloud, platform, solutions and Autonomous Vehicles/Advanced Driver Assistance Systems
- Academic experience:
  - BA, MS, Ph.D., Computer Science, Columbia University.
  - Approx. 15 semesters as an Adjunct Professor.
  - Professor of Professional Practice in CS (2018)
  - Courses:
    - E1006: Intro. to Computing
    - W4111: Intro. to Databases
    - E6998, E6156: Advanced Topics in SW Engineering (Cloud Computing)

I have taught some version of this class 10 times.

- Approx. 65 technical publications; Approx. 12 patents.

Personal:

- Two children:
  - College Sophomore.
  - 2019 Barnard Graduate.
- Hobbies:
  - Krav Maga, Black Belt in Kenpo Karate.
  - 1LT, New York Guard.
  - Bicycling.
  - Astronomy.
  - Languages:
    - Proficient in Spanish.
    - Learning Arabic.

Course Logistics

- Sessions:
  - Lectures:
    - Session is Fridays, 1:10 – 3:40 PM. The format will be:
      - Lecture/presentation from 1:10 to approx. 2:40.
      - Discussion of projects, questions, ... From 2:50 to 3:40.
    - I can cover material much, more quickly than you can implement in projects.
      Typical lecture cadence will be
      - Lecture on new material for two lectures.
      - Followed by one lecture period of project discussions and reviews.
      - You will get experience presenting as projects teams, which is an important "soft skill."
  - Instructor Office Hours:
    - Fridays, 8:30-10:00 AM, 4:00-5:30 (488 CSB)
    - I frequently have extra office hours (online and in-person), usually around due dates for assignments.
- Grading and assignments:
  - This is a team project class. Students form 5 person teams.
  - Final project's completion of objectives determines the grade.
  - There are mandatory weekly project status reports and periodic reviews.

Some Details: Assignments Structure and Objectives

- No exams.
- Projects:
  - We will form small teams and build a simple, realistic microservice/cloud application.
  - The teams will use a simplified version of Agile SW Development.
  - There will be a common core set of functionality to help you get started, e.g. user registration.
  - The teams can choose their project functionality -- what application do you want to build?
  - Teams have some flexibility on which cloud technology they use based on their interests.
- Course objectives:
  - Practical experience with modern technology for building solutions in an agile team.
  - Be able to say that you have built a microservice/cloud application delivering APIs, and cite a list of technology used, e.g. IaaS, PaaS, SaaS, API Management, DBaaS, federated security, ...
  - Become a better programmer through experience with patterns and best practices.
    My experience has been that most students write "crappy code."
  - Prepare you for internships and jobs.
  - Have seriously cool stuff to put on your resume and discuss on interviews.

Course Material

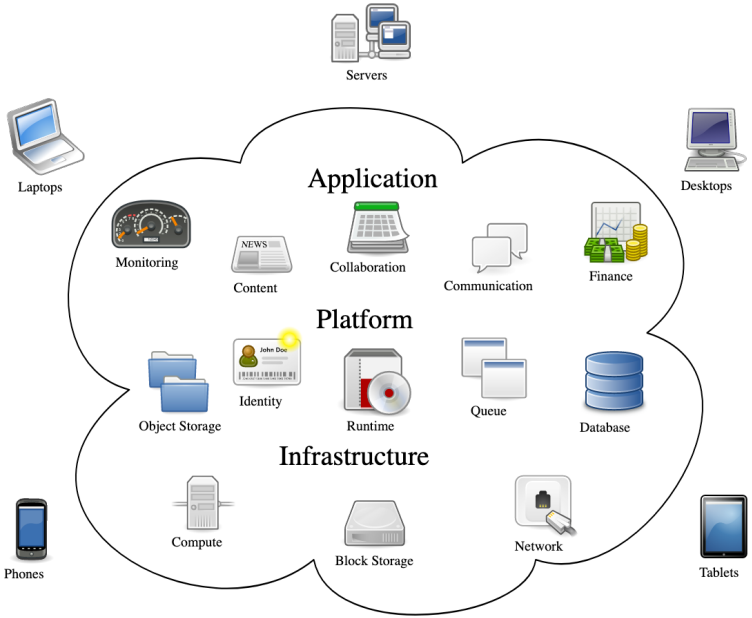
- Course Material:
  - No textbook. Textbooks become out of date in this rapidly changing area. Material would span several books.
  - Lecture notes and code samples on GitHub. Sample code is mostly in Flask/Python and (maybe) JavaScript/Node.js.
    - Course website: https://donald-f-ferguson.github.io/Topics-in-SW-Engineering-F22/
    - Course repo: https://github.com/donald-f-ferguson/Topics-in-SW-Engineering-F22
  - References to web documents and tutorials.
  - There will be some programming and example for UI in HTML/CSS/Angular, but this is not the focus of this course. UI/User Experience is a multi-semester course by itself.
- Environment:
  - Required:
    - AWS (team) free tier account
    - We will try to use Azure and but do not sign up yet.
    - Trello –– project planning, light weight agile development.
  - Recommended:
    - PyCharm (free student licenses https://www.jetbrains.com/student/) and/or PyCharm
    - Ananconda to read Jupyter Notebooks for lectures (https://www.anaconda.com/download/#macos)
    - WebStorm for JavaScript (same link as PyCharm)
- Collaboration environments:
  - Slack: Please join the channel #e6156f21 – https://join.slack.com/t/dff-columbia/shared_invite/zt-1cjbu68ty-RvaI7vLghvo0SFRFP4qbMg
  - We will use the Ed Discussion features, accessible through CourseWorks, for discussions.

Core Concepts

Cloud Computing – Core Concepts

Cloud Computing

- "Cloud computing is an information technology (IT) paradigm that enables ubiquitous access to shared pools of configurable system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility." (https://en.wikipedia.org/wiki/Cloud_computing)



NIST Reference Architecture

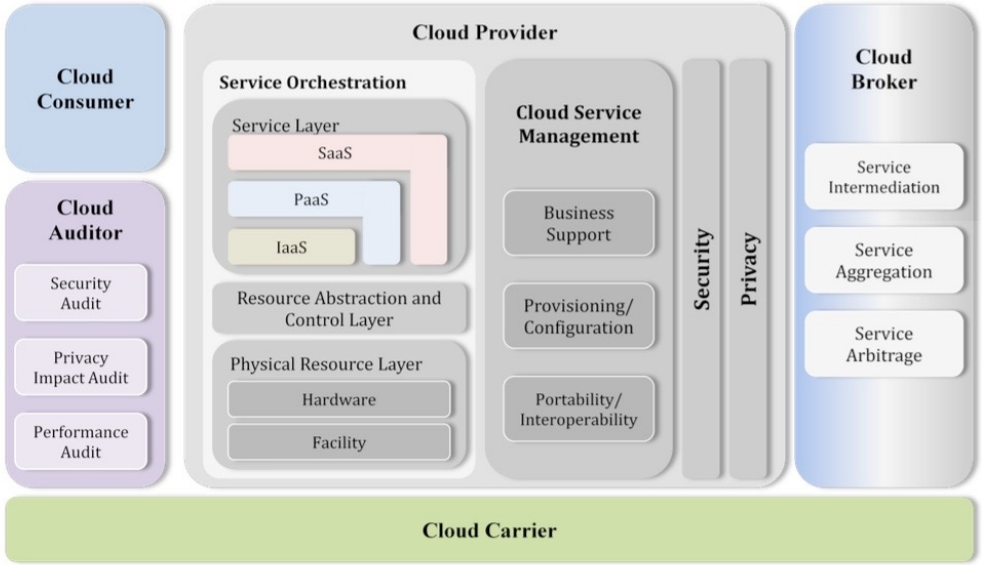


Figure 1: The Conceptual Reference Model

Core Layers

Cloud Computing Layers

Our first cloud elements will be IaaS (VMs).

Resources Managed at each layer

Software as a service (SaaS)

Business Applications, Web Services, Multimedia

Application

- Google Apps
- Salesforce, Zendesk, ServiceNow, Concur, … …
- Office 365, Trello
- … …

Platform as a service (PaaS)

Software Framework (Java, .NET)
Storage (DB, File)

Platform

- Google App Engine, AWS Elastic Beanstalk, …
- AWS SQS, Azure Queuing Service, Google PubSub
- DynamoDB, Cosmos DB, … …
- … …

Infrastructure as a service (IaaS)

Computation (VM) Storage (block)

Infrastructure

- AWS EC2, Google Compute Engine, Azure VMs, … …
- AWS Elastic Container Service, Azure Kubernetes, … …

CPU, Memory ,Disk, Bandwidth

Hardware

- IaaS is compute, storage, networking.
- With an OS.
- Delivered as a set of VMs and/or containers.

Cloud Concepts – One Perspective

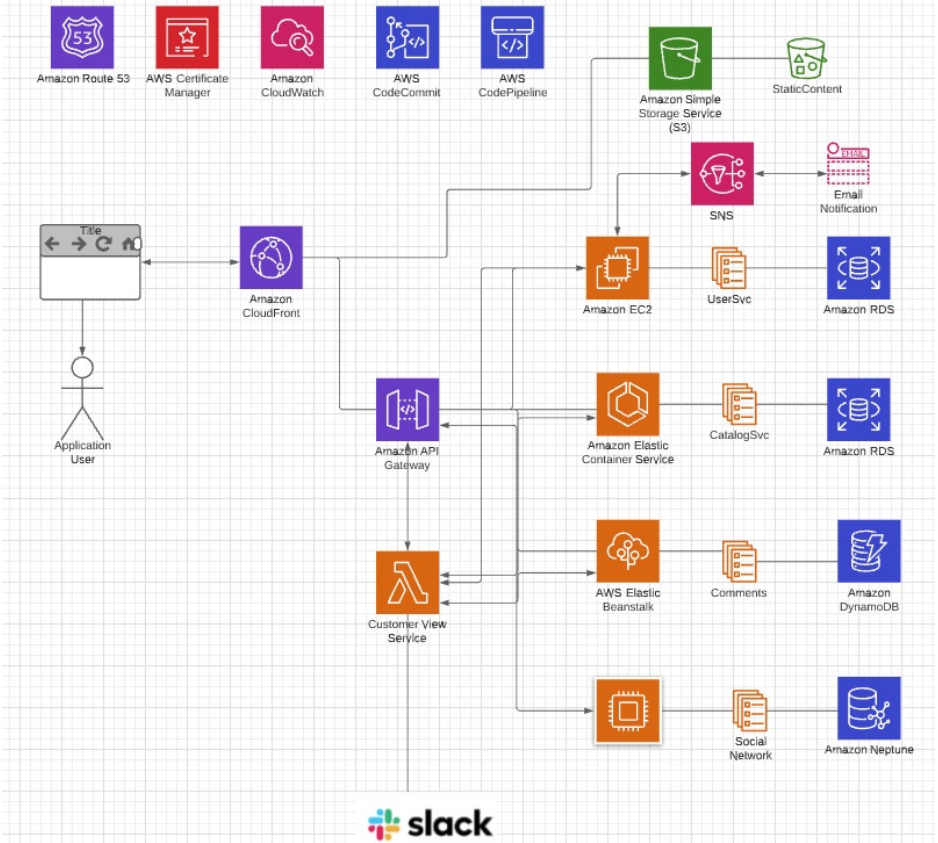
Categorizing and Comparing the Cloud Landscape

http://www.theenterprisearchitect.eu/blog/2013/10/12/the-cloud-landscape-described-categorized-and-compared/

6	**SaaS**	Applications			*End-users*
5	**App Services**	App Services	Communication and Social Services	Data-as-a-Service	*Citizen Developers*
4	**Model-Driven PaaS**	Model-Driven aPaaS, bpmPaaS	Model-Driven iPaaS	Data Analytics, baPaaS	*Rapid Developers*
3	**PaaS**	aPaaS	iPaaS	dbPaaS	*Developers / Coders*
2	**Foundational PaaS**	Application Containers	Routing, Messaging, Orchestration	Object Storage	*DevOps*
1	**Software-Defined Datacenter**	Virtual Machines	Software-Defined Networking (SDN), NFV	Software-Defined Storage (SDS), Block Storage	*Infrastructure Engineers*
0	**Hardware**	Servers	Switches, Routers	Storage	
		Compute	Communicate	Store	

Motivating Sample Application (Topology)

- Note: Cannot show all of the technology and connections with creating spaghetti.
- Databases/Storage:
  - RDS
  - DynamoDB
  - MongoDB (or Document DB)
  - Neptune (or Neo4j)
  - S3
- Networking/Communication:
  - Route 53 (DNS)
  - Certificate Management
  - CloudFront
  - VPC
  - API Gateway
- Compute:
  - EC2
  - Elastic Beanstalk
  - Elastic Container Service (or Docker)
  - Lambda Functions
- Integration/collaboration:
  - SNS, SQS
  - Email
  - Slack
- Continuous Integration/Continuous Deployment:
  - Code Commit, Code Pipeline
  - Or GitHub and actions



Microservices

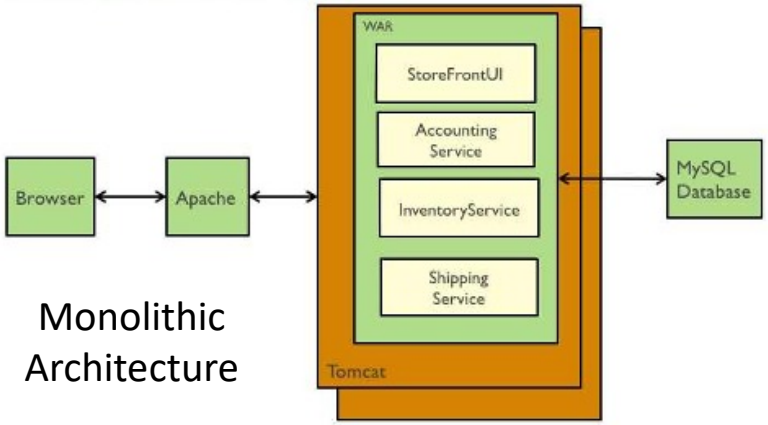
Microservices (https://microservices.io/index.html)

What are microservices?

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

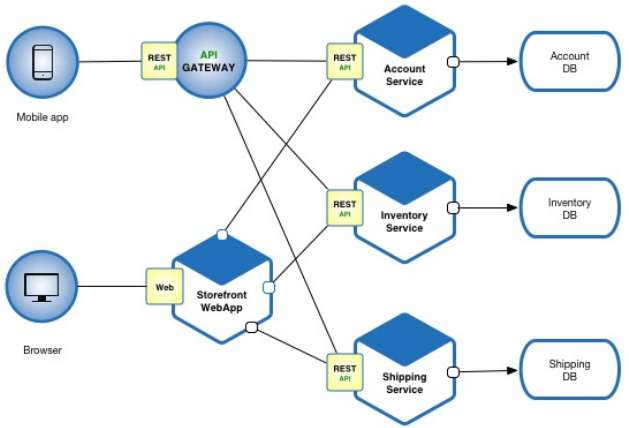
- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.



Monolithic Architecture

Microservices Architecture



Microservice Characteristics

(https://dzone.com/articles/what-is-microservices-an-introduction-to-microserv)

Microservice Characteristics

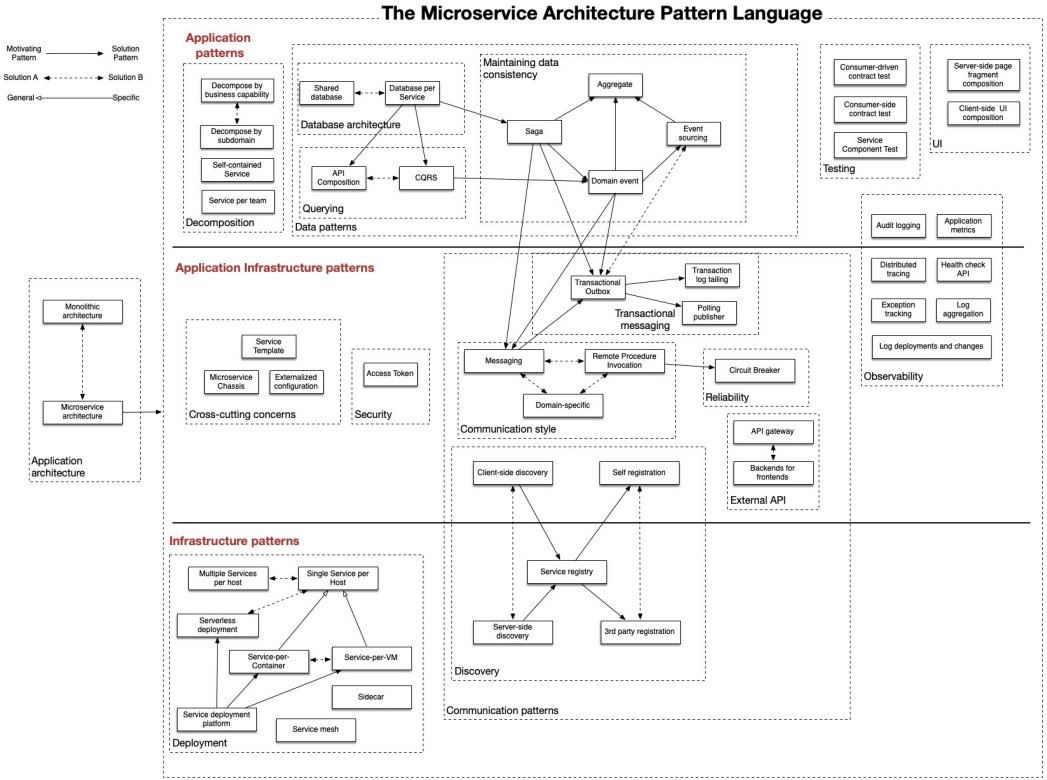
- **Decoupling** - Services within a system are largely decoupled, so the application as a whole can be easily built, altered, and scaled.
- **Componentization** - Microservices are treated as independent components that can be easily replaced and upgraded.
- **Business Capabilities** - Microservices are very simple and focus on a single capability.
- **Autonomy** - Developers and teams can work independently of each other, thus increasing speed.
- **Continous Delivery** - Allows frequent releases of software through systematic automation of software creation, testing, and approval.
- **Responsibility** - Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible.
- **Decentralized Governance** - The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems.
- **Agility** - Microservices support agile development. Any new feature can be quickly developed and discarded again.

Design Patterns

- "In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system." (https://en.wikipedia.org/wiki/Software_design_pattern)

- We will cover design patterns in several aspects:
  - SW patterns for building code "in microservices." How do you write the code?
  - Architecture patterns for microservices and composition into complex solutions.
  - Cloud architecture patterns: how do you solve problems with cloud technology.

Microservice Pattern Language (One Perspective)



https://microservices.io/patterns/index.html

Microservice Details and Perspectives

- Some sites for good, introductory overviews and more information about microservices:
  - https://martinfowler.com/articles/microservices.html
  - https://microservices.io/
  - https://aws.amazon.com/microservices/
- What is true of any:
  - Mapping, classification, taxonomy, ...?
  - Technology definition, ...?
  - Enumeration of technology pros and cons ... ...?
- Answers:
  - "... ... it is ... ...
    More honor'd in the breach than the observance"
  - If you have three subject matter experts, you have 7 conflicting opinions.
  - "The most dangerous thing in the world is a 2nd LT with a map."
    Like any map, it is both a way of orientating yourself and a way of having a spectacular disaster because you are studying the map instead of thinking.

Full Stack Application Introduction

Full Stack Application

Full Stack Developer Meaning & Definition

In technology development, full stack refers to an entire computer system or application from the front end to the back end and the code that connects the two. The back end of a computer system encompasses "behind-the-scenes" technologies such as the database and operating system. The front end is the user interface (UI). This end-to-end system requires many ancillary technologies such as the network, hardware, load balancers, and firewalls.

FULL STACK WEB DEVELOPERS

Full stack is most commonly used when referring to web developers. A full stack web developer works with both the front and back end of a website or application. They are proficient in both front-end and back-end languages and frameworks, as well as server, network, and hosting environments.

Full-stack developers need to be proficient in languages used for front-end development such as HTML, CSS, JavaScript, and third-party libraries and extensions for Web development such as JQuery, SASS, and REACT. Mastery of these front-end programming languages will need to be combined with knowledge of UI design as well as customer experience design for creating optimal front-facing websites and applications.

https://www.webopedia.com/definitions/full-stack/

Full Stack Web Developer

A full stack web developer is a person who can develop both **client** and **server** software.

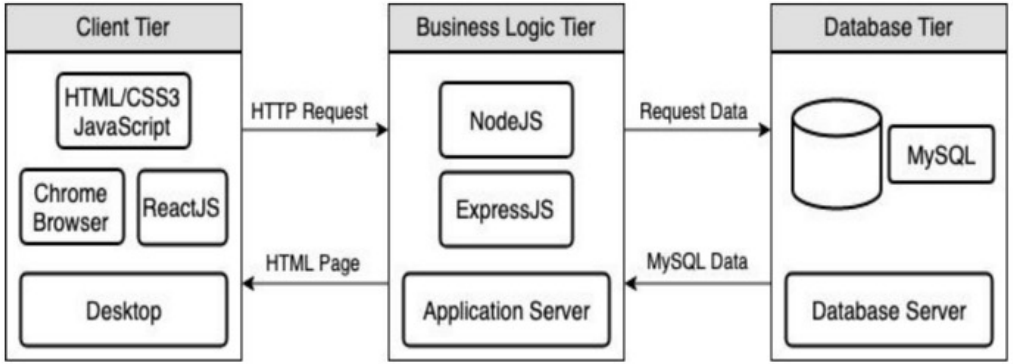
In addition to mastering HTML and CSS, he/she also knows how to:

- Program a **browser** (like using JavaScript, jQuery, Angular, or Vue)
- Program a **server** (like using PHP, ASP, Python, or Node)
- Program a **database** (like using SQL, SQLite, or MongoDB)

https://www.w3schools.com/whatis/whatis_fullstack.asp

- There are courses that cover topics:
  - COMS W4156: Advanced Software Engineering
  - COMS W4111: Introduction to Databases
  - COMS W4170 - User Interface Design
- This course will focus on cloud realization, microservices and application patterns, … …
- Also, I am not great at UIs … … We will not emphasize or require a lot of UI work.

Full Stack Web Application

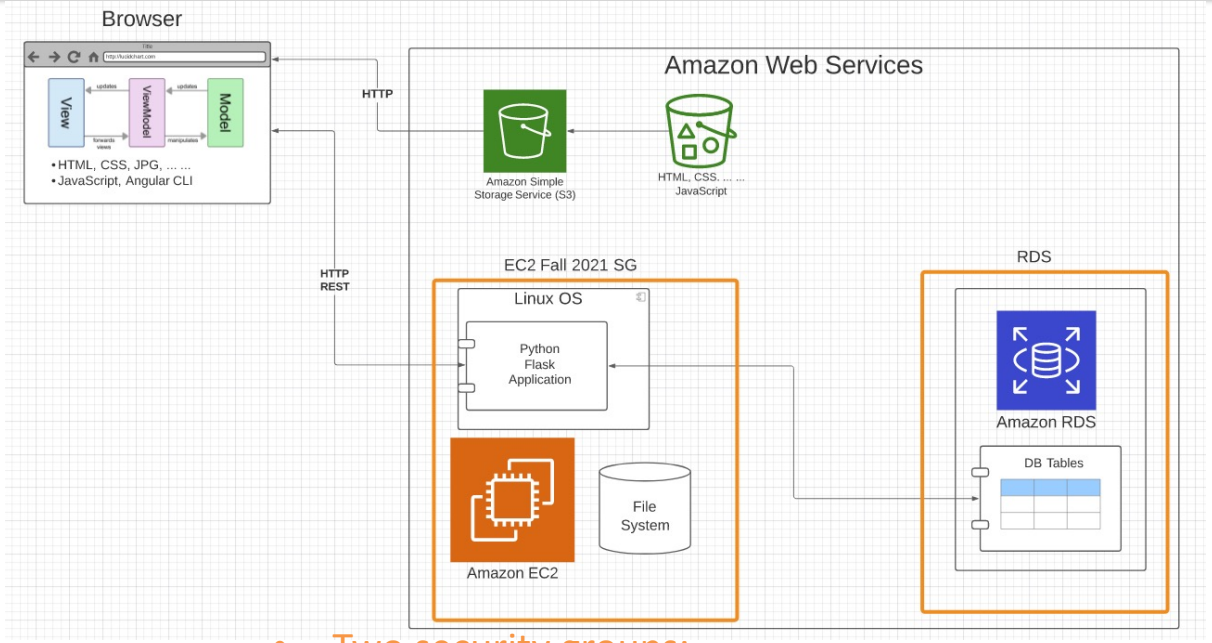


https://levelup.gitconnected.com/a-complete-guide-build-a-scalable-3-tier-architecture-with-mern-stack-es6-ca129d7df805

- My preferences are to replace React with Angular, and Node with Flask.
- There are three projects to design, develop, test, deploy, ...
  1. Browser UI application.
  2. Microservice.
  3. Database.
- We will initial have two deployments: local machine, virtual machine/S3/RDS.

Sample Application Structure

- Browser Application:
  - Content: HTML, CSS, … …
  - MVVM with Angular CLI
- Static Content:
  - S3 Bucket
  - Web site hosting
- Server: Amazon EC2 instance (IaaS)
  - Linux
  - Web Application Server
    - Flask
    - Python
- Database: Amazon RDS (DBaaS)
  - Simple tables
  - MySQL Database Server
- Lifecycle:
  - Develop and demo locally
  - Deploy and configure onto AWS
  - Will work on CI/CD later.

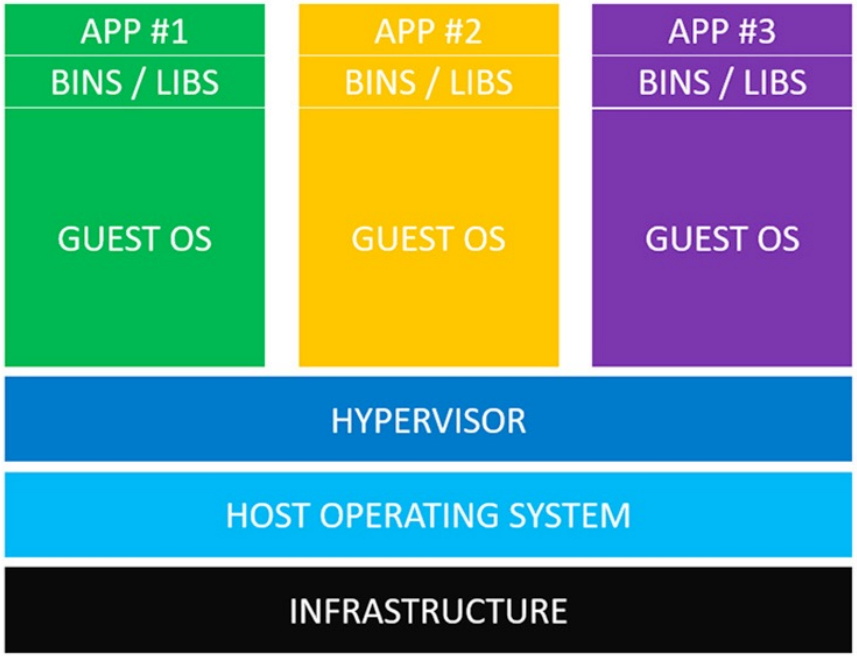


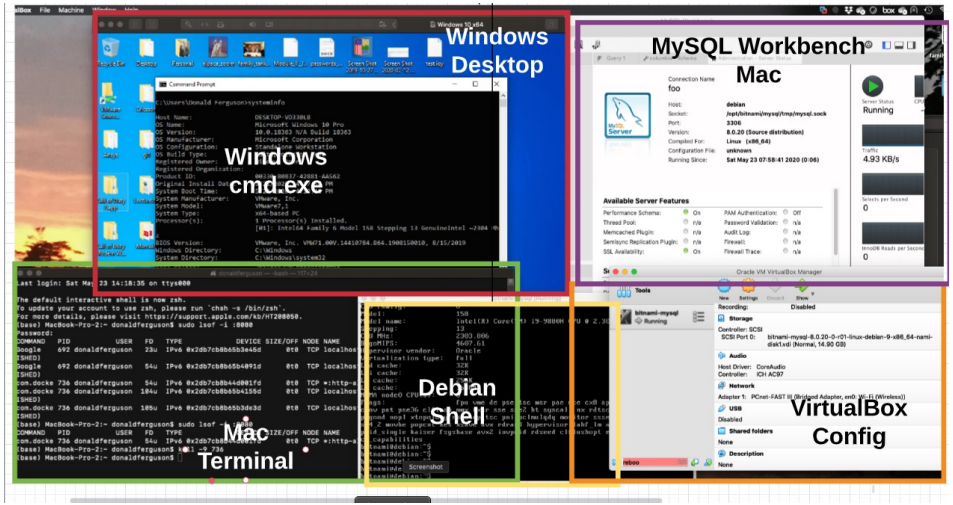
- Two security groups:
  - One for EC2 instance
  - One for RDS
- Slightly more secure than 1 SG.
- Will cover security later.

VMs, IaaS

Virtualization, Virtual Machines

- "In computing, a virtual machine (VM) is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination." (https://en.wikipedia.org/wiki/Virtual_machine)

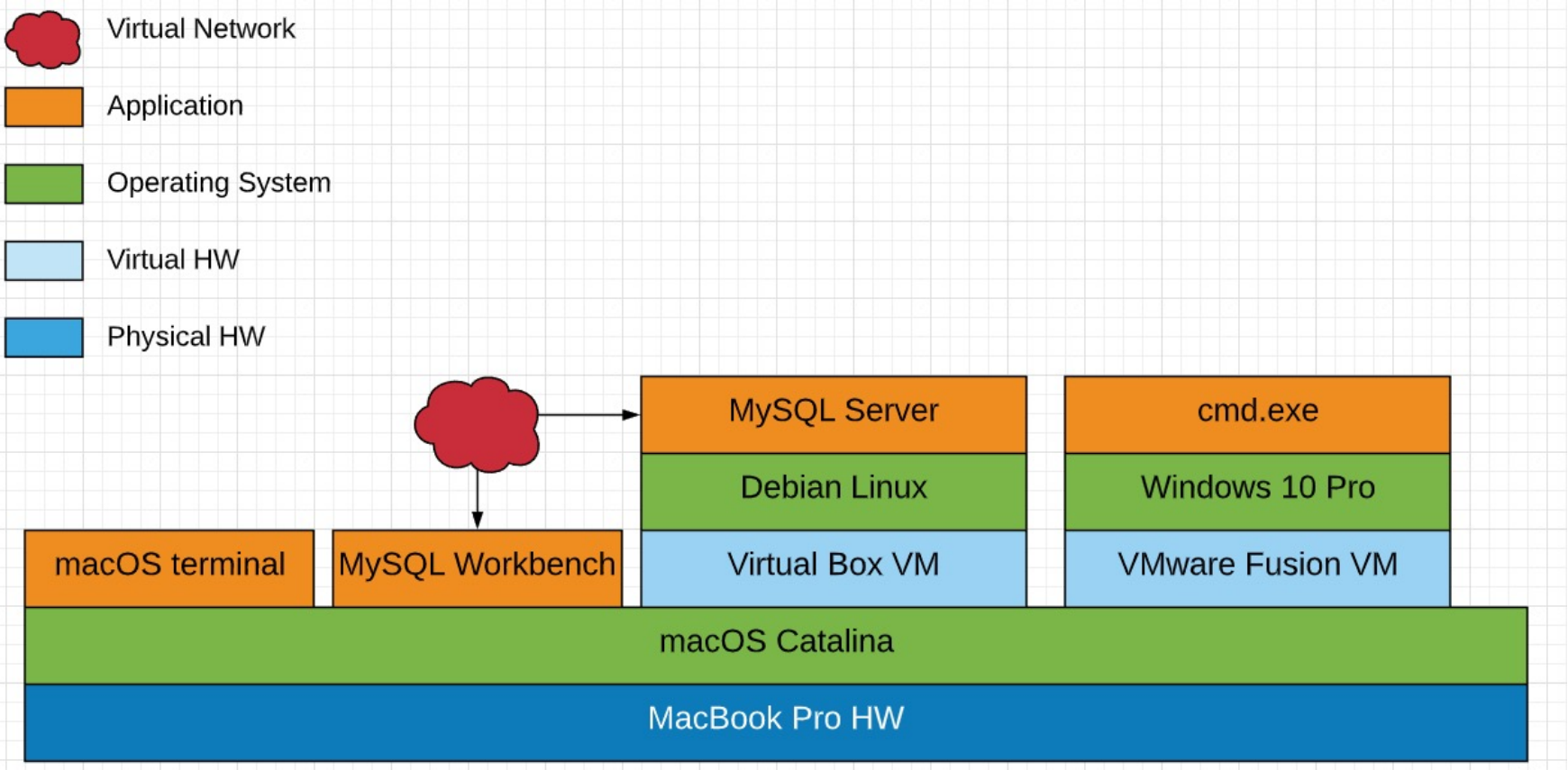




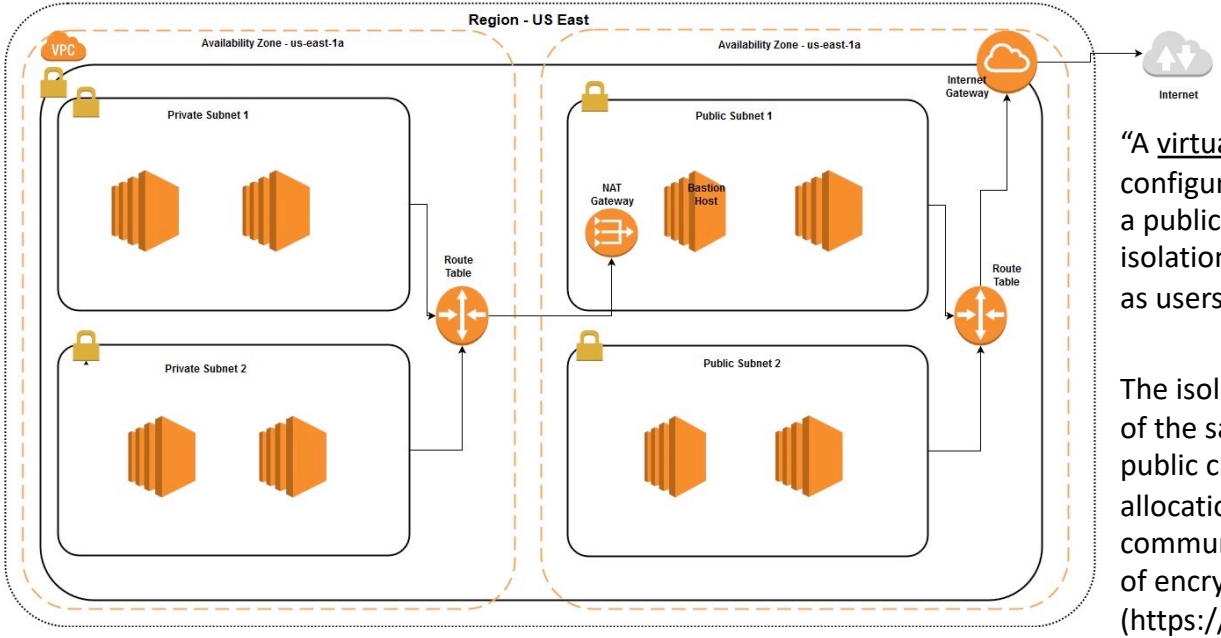
On mac laptop, I have:

- One host operating system and HW (Mac, MacOS)
- Two hypervisors (Virtual Box, VMWare Fusion)
- Two guest OS: Windows 10, Debian Linux

Virtual Machines on Physical Machines



Virtual Private Cloud



"A virtual private cloud (VPC) is an on-demand configurable pool of shared resources allocated within a public cloud environment, providing a certain level of isolation between the different organizations (denoted as users hereafter) using the resources. ... ...

The isolation between one VPC user and all other users of the same cloud (other VPC users as well as other public cloud users) is achieved normally through allocation of a private IP subnet and a virtual communication construct (such as a VLAN or a set of encrypted communication channels) per user." (https://en.wikipedia.org/wiki/Virtual_private_cloud)

- A virtual machine on my laptop has a virtual adaptor connected to a real network.
- A virtual private cloud is a set of virtual machines connected to a virtual network.
- All cloud providers support virtual private clouds with the same concepts, but with slightly different realizations and terms.

Some Terminology (AWS)

There are several online tutorials. Understanding some terms is useful.

- AWS has the concept of a Region, which is a physical location around the world where we cluster data centers. We call each group of logical data centers an Availability Zone. Each AWS Region consists of multiple, isolated, and physically separate AZ's within a geographic area.
- An Availability Zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. AZs give customers the ability to operate production applications and databases that are more highly available, fault tolerant, and scalable than would be possible from a single data center.
- A VPC is a virtual private cloud, which works like a private network to isolate the resources within it.
- A route table contains a set of rules, called routes, that are used to determine where network traffic is directed.
- A subnet is a defined set of network IP addresses that are used to increase the security and efficiency of network communications. You can think of them like postal codes, used for routing packages from one location to another.
- A network interface represents a virtual network card. The network interface displays its network interface ID, subnet ID, VPC ID, security group, and the Availability Zone that it exists in.
- A security group is a set of rules that controls the network access to the resources it is associated with. Access is permitted only to and from the components defined in the security group's inbound and outbound rules.
- An internet gateway is a VPC component that allows communication between instances in your VPC and the internet.
- A VPC endpoint enables you to privately connect your VPC to supported AWS services without using public IP addresses.

All clouds have similar concepts but with different names.

Let's Build our First VPC

- Well, actually, let's not do that for now.
- We are just going to create our first virtual machine.
- But,
  - The virtual machine is in my "default" VPC.
  - Setting up and using the VM requires understanding some of the concepts.
- We can mostly just use the EC2 wizard for now.
- I am going to set up a VM to host our first cloud application
  - I will deploy the application and supporting SW.
  - Basic tasks:
    1. Configure and deploy VM, SSH credentials, security group rules.
    2. Install supporting software: Python, MySQL.
    3. Set up the database and deploy the application.
    4. Run "Hello World" test.
- <u>But first, let's walk through the application structure and implementation.</u>

Assignment/Project 1

Assignment 1/Project Step 1

- Individual:
  - Create an AWS account.
  - Configure laptop
    - IDE:
      - Get JetBrains student license.
      - Install PyCharm
      - Optional:
        - » Install WebStorm
        - » Install DataGrip
    - Install Anaconda (new environment)
    - Install MySQL
    - Postman
  - Clone and test starter application:
    - UI: https://github.com/donald-f-ferguson/F22-Simple-Web-UI.git
    - Microservice: https://github.com/donald-f-ferguson/F22-Starter-Microservice.git
    - DB: https://github.com/donald-f-ferguson/F22-Starter-DB.git
- Group: Form 5-7 student team.
  - Each team must have a designated communication focal point. The professor and TAs will use to communicate with teams.
  - The focal point must enter the information on the Google form for the team: https://forms.gle/t1wu8a3DZPJYnEks5
  - Create AWS group account for team and add members.
  - Create a GitHub Project for the course, and add team members: https://docs.github.com/en/issues/planning-and-tracking-with-projects/creating-projects/creating-a-project